Arithmetic Complexity - A Survey

Lecturer: Avi Wigderson * Scribe: Oded Regev †

February 7, 2002

1 Introduction

We describe the state of the art in the computational complexity of natural polynomials, e.g. symmetric functions, determinant, matrix multiplication. The model we consider is that of arithmetic circuits over infinite fields which is believed to be easier to handle than that of finite fields or of the Boolean ring. We present some techniques, results and open problems. The interested reader is referred to a book by Bürgisser et al. [4] and to two survey papers [24, 28].

2 Preliminaries and Upper Bounds

We only consider computation of polynomials over fields $\mathbb{F}$ of $ch(\mathbb{F}) = 0$ which will be either the rationals $\mathbb{Q}$ or the complex numbers $\mathbb{C}$. Before we formally define the model, let us begin with a simple example. Assume we want to compute the univariate polynomial x^d. Let $S(x^d)$ be the minimal size of such a computation. Then, $S(x^d) \sim \log d$ because we can multiply x by itself and then multiply the result by itself and so on. After $\lfloor \log d \rfloor$ steps we have all the polynomials $x, x^2, x^4, \ldots, x^{2^{\lfloor \log d \rfloor}}$ and now we can multiply a subset of them in order to reach x^d. The total number of steps is at most $2 \log d$, or in other words, $S(x^d) \leq 2 \log d$.

Definition 2.1 *Given n input polynomials $x_1, x_2, \ldots x_n$, a computation (or a circuit) of size s of the k polynomials $y_1, y_2, \ldots y_k$ is a sequence $g_1, g_2, \ldots, g_s$ where $g_i = x_i$ for $i = 1, \ldots, n$ and $g_{n-k-1+i} = y_i$ for $i = 1, \ldots, k$. The polynomial g_k for $k > n$ is the result of one of the following operations over two polynomials g_i, g_j where $i < j < k$. It can either be a linear combination $g_k = \alpha g_i + \beta g_j$ for $\alpha, \beta \in \mathbb{F}$ or a multiplication $g_k = g_i \cdot g_j$. We denote by $S(P)$ the size of the shortest calculation of a set of polynomials P.*

*Institute for Advanced Study, Princeton, NJ. E-Mail: avi@ias.edu.
†Institute for Advanced Study, Princeton, NJ. E-Mail: odedr@ias.edu.

Note that we define the size as the number of operations and we are not interested in the size of the polynomials themselves.

Definition 2.2 *The depth of a computation is the longest path between an input and an output in the tree underlying the computation. We denote by $D(P)$ the minimal depth of a computation of P.*

The measure of depth is interesting since a low depth indicates a highly parallel computation. In the computation of the univariate polynomial x^d described above, the depth is around $\log d$ and therefore $D(x^d) \leq O(\log d)$. By considering the degree of the intermediate polynomials, it can be seen that both the size and the depth of a computation of x^d cannot be much lower than $\log d$.

Open Problem 2.3 *What is $S(\Pi_{i=1}^{d}(x-i))$?*

It can be seen that if $S(\Pi_{i=1}^{d}(x-i)) = (\log d)^{O(1)}$ then factoring integers is in $P/poly$.

Linear Operators: We now consider the computation of $A\bar{x}$ where A is a fixed $n \times n$ matrix and $\bar{x} = (x_1, x_2, \ldots, x_n)$. That is, we want to find $\bar{y} = A\bar{x}$, or, $y_i = \langle \bar{a}_i, \bar{x} \rangle = \sum_j a_{ij} x_j$. It can be shown that there exists A such that $S(A) \geq c \cdot n^2$ (actually, almost all matrices A satisfy this property). However,

Open Problem 2.4 *Find an* explicit *matrix A such that $S(A) \gg n$ (even $S(A) > 3n$ is interesting).*

For certain matrices this computation is known to be easier. For example, consider the Discrete Fourier Transform matrix:

$$DFT = \begin{pmatrix} 1 & \omega & \omega^2 & \ldots \\ 1 & \omega^2 & \omega^4 & \ldots \\ \vdots & & & \end{pmatrix}$$

where $\omega = e^{2\pi i/n}$ in the nth complex root. Here, $S(DFT) \leq O(n \log n)$ by using the computation known as the Fast Fourier Transform [5]. It was shown by Morgenstern [14] that this is tight assuming that the addition operations in the computation are limited to $g_k = \alpha g_i + \beta g_j$ with $|\alpha|, |\beta| \leq 1$.

The DFT matrix is a special case of the Vandermonde matrix:

$$V(\bar{z}) = \begin{pmatrix} 1 & z_1 & z_1^2 & \ldots \\ 1 & z_2 & z_2^2 & \ldots \\ \vdots & & & \end{pmatrix}$$

where $\bar{z} = (z_1, z_2, \ldots, z_n)$. The Vandermonde transformation can be computed in $O(n \log^2 n)$ operations [1, 3, 4]. Shoup and Smolensky [19] showed that at least $\Omega(n \log n)$

operations are required to compute the Vandermonde matrix with the assumption that the computation is of depth $O(\log n)$ (a similar lower bound applies to any poly-logarithmic depth circuit). One should note that their lower bound assumes that the values $z_1, \ldots, z_n$ are algebraically independent. That is, the circuit is assumed not to use any relations between the values $z_1, \ldots, z_n$. Notice that a trivial $\Omega(n^2)$ lower bound holds if the matrix contains n^2 algebraically independent values. Nevertheless, their result is interesting because they show an $\Omega(n \log n)$ lower bound with only n algebraically independent values.

Open Problem 2.5 *Show that $S(V(\bar{z})) = \Omega(n \log n)$ (with no assumption on the depth).*

Multiplication of Polynomials: Consider two polynomials $x = x(t) = \sum_i x_i t^i$ and $y = y(t) = \sum_i y_i t^i$ given as a sequence of coefficients $x_0, x_1, \ldots, x_d, y_0, y_1, \ldots, y_d$ where d denotes the degree. Adding the two polynomial can be done with exactly $d+1$ addition operations, $S(x+y) = d+1$. But what about computing the multiplication xy ? It can be done with d^2 operations by computing each coefficient separately. We can improve it as follows. Choose any $2d+1$ values $a_0, \ldots, a_{2d}$ and calculate $x(a_0), \ldots, x(a_{2d}), y(a_0), \ldots, y(a_{2d})$. By performing $2d+1$ multiplications, we obtain an evaluation of $x \cdot y$ in these $2d+1$ values, $(x \cdot y)(a_i) = x(a_i) \cdot y(a_i)$. Now, since $x \cdot y$ is a polynomial of degree $2d$, we can interpolate $x \cdot y$ from its evaluation in $2d+1$ values. That is, finding each coefficient requires the addition of the $2d+1$ evaluations with an appropriate choice of coefficients.

The computation described above is a three step process. First, we changed the representation of the polynomials x and y. Then we multiplied them and finally we returned to the original coefficient representation. Computing the multiplication in the second step is very fast, only $2d+1$ multiplications. Changing the representations, however, can take $O(d^2)$ operations because each evaluation operation requires d operations. This can be improved by choosing the points $a_j = e^{2\pi i j/(2d+1)}$ as our evaluation points. Now, changing the representation takes only $d \log d$ by using the Fast Fourier Transform and we conclude that $S(x \cdot y) \leq d \log d$.

Matrix Multiplication: Given two $n \times n$ matrices, X, Y, what is $S(X \cdot Y)$? Obviously, the product of two matrices can be computed in n^3 steps. After many attempts to show that n^3 is optimal had failed, Strassen [21] showed a more efficient way of calculating the product. The idea is the following. Given two 2×2 matrices

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x & y \\ z & w \end{pmatrix},$$

the multiplication of them can be computed with 7 multiplications instead of 8 (though at the cost of more additions). The same method can be applied recursively by representing each $n \times n$ matrix as four $\frac{n}{2} \times \frac{n}{2}$ matrix. The resulting computation is of size $O(n^{\log_2 7}) <$

$O(n^{2.81})$. Currently, the best bound is $O(n^{2.376\ldots})$ (due to Coppersmith and Winograd [6]).

Open Problem 2.6 *What is the best bound on the computation of matrix multiplication? Is it $O(n^2)$?*

Direct Sum: An interesting corollary of the improved matrix multiplication computation is the following. Given n vectors $\bar{x}^1, \bar{x}^2, \ldots, \bar{x}^n$ and a fixed $n \times n$ matrix A, we want to compute $A\bar{x}^1, \ldots, A\bar{x}^n$. By performing each calculation separately using the computation described earlier we obtain an $O(n^3)$ computation. Notice however that this computation is actually a matrix multiplication and can therefore be performed using only $S(X \cdot Y)$ arithmetic operations. Therefore, the cost of performing n identical computations on n different inputs is less than n times the cost of performing just one computation. In other words, the "direct sum" condition fails for arithmetic circuits.

Verification of Matrix Multiplication: A related problem is the following 'decision' problem: given X, Y and Z, does Z equal $X \cdot Y$? It is not known whether this problem is easier. With the use of randomization, however, this decision problem can be solved with high probability in $O(n^2)$ [7]. The details of the proof and the following two simple exercises are left to the reader.

Exercise 2.7 *Show that squaring a matrix is as hard as multiplying two matrices.*

Exercise 2.8 *Show that $S(X_1 \cdot X_2 \cdot \ldots \cdot X_d) \leq S(X \cdot Y) \cdot d$ and $S(X^d) \leq S(X \cdot Y) \cdot \log d$.*

Determinant: Computing the determinant of a matrix can be performed in $O(n^3)$ arithmetic steps. The idea is to use a Gaussian elimination which brings the matrix to a lower triangular form without changing its determinant. Computing the determinant of a lower triangular form consists of simply multiplying the elements of the diagonal. The Gaussian elimination should be done without divisions since division is not an elementary operation. The details are left to the reader as an exercise. We can also use the following result of Strassen:

Theorem 2.9 ([23]) *Any computation that uses divisions can be transformed into a computation that does not use divisions with a polynomial increase in size.*

The best known computation of the determinant is based on matrix multiplication and has size $S(X \cdot Y)$ [27].

Another interesting aspect of computing the determinant is the shortest depth of the computation. Recall that the depth is the longest path from an input to an output. By using the method outlined above, the depth is $D(DET_n) \leq O(n^2)$. The best known result is that $D(DET_n) \leq (\log n)^2$ [9, 26]. Moreover, this computation is still done with $S(X \cdot Y)$ arithmetic operations.

This result is a special case of a more general result that relates the depth of computing any polynomial to its degree:

Theorem 2.10 ([26]) *Any polynomial size circuit that computes a multivariate polynomial f can be transformed into an equivalent circuit of depth at most* $(\log S(f))\cdot\log(degf)$.

For example, this result translates to $D(DET_n) \leq O((\log n)^2)$ since the determinant is a polynomial of degree n and $S(DET_n)$ is polynomial. A similar relation is not believed to hold in the model of computation over finite fields.

3 Lower Bounds

3.1 Restricted Models of Computation

Monotone Circuits: Many lower bounds for arithmetic circuit complexity are based on some assumptions that restrict the power of the computation. For example, we have already mentioned that not allowing divisions does not make the computation much longer. We might also consider *monotone* computations. These are computations that are not allowed to use negative coefficients (this makes sense when $\mathbb{F} = \mathbb{Q}$ or $\mathbb{F} = \mathbb{R}$). That is, the addition operation is only $g_k = \alpha g_i + \beta g_j$ where α, β are two positive reals. Note that monotone computations are limited to polynomials whose coefficients are positive. Exponential lower bounds are known in the model of monotone circuits for some explicit functions such as the permanent of a matrix. Namely, Jerrum and Snir [10] (also [15]) have shown that $S^M(Per_n) \geq 2^n$ where S^M denotes the size of the shortest monotone computation. There are also some lower bounds on the depth of a monotone computation. For example, Shamir and Snir show in [18] that $D^M(X_1 \cdot X_2 \cdot \ldots \cdot X_d) \geq (\log n)(\log d)$ for d $n \times n$ matrices. This implies a similar lower bound for $D^M(X^d)$ since a simple argument shows that multiplying d matrices can be done by calculating X^d for a certain matrix X. A lower bound of $D^M(Sym_n^d) \geq (\log n)(\log d)$ is shown in [18] where $Sym_n^d = \sum_{|S|=d, S\subseteq[n]} \Pi_{x\in S} x$. This lower bound does not hold in the non-monotone case since there exists a depth $\log n$ computation of Sym_n^d (this is implied by a result of Ben-Or that appears later).

Non-commutative Circuits: Another possible restriction is the non-commutative model. Here, we assume that our variables are not commutative, that is, $x_1 \cdot x_2$ does not necessary equal $x_2 \cdot x_1$. This model is considered in a paper by Nisan [15]. For example, it is shown that $D^N(Det_n) = \Omega(n)$ where $D^N(Det_n)$ denotes the minimum depth of a non-commutative computation of the determinant (with some order on the variables in the monomials).

Open Problem 3.1 *Find an explicit function for which there exists no polynomial size computation in the non-commutative model.*

3.2 The General Model

In the following, we consider lower bounds without any restrictions on the type of operations allowed. All known lower bounds are based on the idea of a progress measure. A progress measure is a function that is applied to intermediate states of the computation. While we will see different progress measures, they all share the same three properties. The initial value (of inputs) is low, the final value (of outputs) is high, and the increase by each elementary operation is low. The method of progress measures is used in virtually all the lower bounds for both the arithmetic and the Boolean computation complexity. The limited power of this method was shown by Razborov and Rudich [17]. Namely, they show that this method cannot prove superpolynomial lower bounds for general circuits in the Boolean case (assuming one-way functions exist). An interesting open questions is whether a similar result holds for the arithmetic case.

Algebraic Geometry: The lower bound we consider now is based on algebraic geometry. We assume we operate over the complex numbers. We define the degree of a set of polynomials $h_1, \ldots, h_k$ as the number of affine linear forms $l_1, \ldots, l_t$ that must be added so as to maximize the finite number of solutions to the equations $h_1 = c_1, \ldots, h_k = c_k, l_1 = 0, \ldots, l_t = 0$ where $c_1, \ldots, c_k$ are any constants. For example, the degree of $y - x^2 + 1$ is 2 since by adding $y = 0$ we have a finite number of solutions, i.e., two solutions. The polynomial $y - x^2$ can be seen to have the same degree by adding the affine linear form $y = 1$. The polynomial $y - x^d$ has degree d since we can add $y = 1$ and the d solutions are $x = 1, e^{2\pi i/d}, \ldots, e^{2\pi i(d-1)/d}$. Similarly, the set of polynomials $\{y_i - x_i^d\}_{i=1}^n$ has degree d^n since we can add $y_i = 1$ for all i and a solution is given by choosing one of the d roots of unity to each of the x_i.

The lower bound we show is $S(x_1^d, x_2^d, \ldots, x_n^d) \geq n \log d$. Notice that it is essentially tight. By using the last example of the previous paragraph, we see that the degree of the output polynomials is n^d. In addition, $deg(x_1, x_2, \ldots, x_n)$ is 1. We only have to show that the progress made by the computation in each step is small. Let $G_t = \{g_1, g_2, \ldots, g_t\}$ be the set of t first polynomials in the computation. If $g_{t+1} = \alpha g_j + \beta g_k$ then $deg(G_{t+1}) = deg(G_t)$ because an assignment to the variables is a solution to G_t iff it is a solution to G_{t+1}. When multiplying two polynomials, a corollary of Bezout's theorem [8, I.7, Thm. 7.7] says that $deg(G_{t+1}) \leq 2deg(G_t)$. Therefore, the progress measure increases by a factor of at most 2 in each step and hence the size of the computation is at least $\log deg(x_1^d, x_2^d, \ldots, x_n^d) = n \log d$. In general,

Theorem 3.2 ([22]) $S(f_1, \ldots, f_k) \geq \log deg(f_1, \ldots, f_k)$

This lower bound is tight for many important set of polynomials. Note that by itself, this method cannot yield a non-trivial lower bound for a single polynomial.

Partial Derivatives: Assume we are given a computation of a multivariate polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_m]$ of degree n and we want to compute the partial derivative $\frac{\partial f}{\partial x_1}$ as

well. This can be done by computing the partial derivative of every polynomial g_i along the computation. For example, if $g_6 = g_5 \cdot g_4$ then we compute the partial derivative of g_6 as $\frac{\partial g_6}{\partial x_1} = \frac{\partial g_5}{\partial x_1} \cdot g_4 + \frac{\partial g_4}{\partial x_1} \cdot g_5$. The size of the computation increases by a constant factor.

Following the same method, we can calculate all m partial derivatives with an increase in size by a factor of m. Surprisingly, Baur and Strassen showed that this can be done without increasing the size of the computation by more than a constant factor:

Theorem 3.3 ([2]) $S(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}) \leq 5S(f)$

A simpler proof can be found in [13]. An improved result that does not increase the depth of the computation by more than a constant factor can be found in [11].

Proof Sketch: Consider the computation of f as a sequence of polynomials $g_1, \ldots, g_s$ where $g_s = f$ is the output of the computation and $g_1, \ldots, g_n$ are the inputs. The computation of all the partial derivatives of f is performed as follows. Assume we can compute $\frac{\partial f}{\partial g_i}$ given $\frac{\partial f}{\partial g_{i+1}}, \ldots, \frac{\partial f}{\partial g_s}$. Then, we append the computation of $\frac{\partial f}{\partial g_{s-1}}, \frac{\partial f}{\partial g_{s-2}}, \ldots, \frac{\partial f}{\partial g_1}$ to the computation of f. The last n polynomials in this sequence are the partial derivatives of f.

We claim that given $\frac{\partial f}{\partial g_{i+1}}, \ldots, \frac{\partial f}{\partial g_s}$ we can compute $\frac{\partial f}{\partial g_i}$. Consider all the appearances of g_i in the computation of f. Say it appears in $g_{m_1}, \ldots, g_{m_k}$ where $m_1, \ldots, m_k \in \{i+1, \ldots, s\}$. Then,

$$\frac{\partial f}{\partial g_i} = \sum_{j=1}^{k} \frac{\partial f}{\partial g_{m_j}} \frac{\partial g_{m_j}}{\partial g_i}.$$

According to our assumption, $\frac{\partial f}{\partial g_{m_j}}$ is given. Computing $\frac{\partial g_{m_j}}{\partial g_i}$ is easy: if g_{m_j} is an addition of the form $\alpha g_i + \beta g'$ then the partial derivative is α. Otherwise, if g_{m_j} is a multiplication of g_i and g' then the partial derivative is g'.

Notice that the number of operations performed while computing the partial derivative with respect to a polynomial g_i is at most twice the number of times g_i is used in the original computation after its computation (that is, its "fan-out"). The sum of the fan-out of all the intermediate polynomials is the number of edges in the computation tree which is at most twice the size of the original computation. Therefore, the total number of operations is at most $4+1 = 5$ times the number of operations in the original computation. ■

The importance of this result is that it allows us to derive many lower bounds on computing *single* polynomials. For example, consider $S(x_1^d + x_2^d + \ldots + x_n^d)$. This function might be considered simpler than $S(x_1^d, x_2^d, \ldots, x_n^d)$ since instead of computing each of the powers separately we only have to compute the sum of them. The result of the previous paragraphs, however, shows that $S(x_1^d + x_2^d + \ldots + x_n^d) \geq \frac{1}{5} S(x_1^{d-1}, x_2^{d-1}, \ldots, x_n^{d-1})$ which is at least $n \log d$ as we have previously seen. Another surprising corollary is that inversion is not harder than the determinant. Namely, $\frac{1}{5} S(X^{-1}) \leq S(Det_n)$ because according to

Cramer's rule, $X^{-1}_{ij} = \frac{\partial DetX}{\partial x_{ij}} \cdot \frac{1}{DetX}$. We can also lower bound the size of a computation of a bilinear form. Namely, $S(y^tAx) \geq \frac{1}{5}S(Ax)$ because the vector Ax is the vector of the partial derivatives of y^tAx according to $y_1, \ldots, y_n$.

Open Problem 3.4 *Prove (or disprove) that* $S(y^tAx) = \Omega(S(Ax))$ *over a* finite *field.*

The result in the previous paragraphs shows that the size of computing the first partial derivatives of a multivariate polynomial is at most a constant factor more than that of the multivariate polynomial itself. Naturally, we also consider the second order partial derivatives.

Open Problem 3.5 *Let* $f(x_1, \ldots, x_n, y_1, \ldots, y_m)$ *be a function of* $n+m$ *variables. Is it true that* $S(\frac{\partial f}{\partial x_i \partial x_j}) \leq O(S(f) + n^2)$ *?*

An exercise to the reader is to show that if the statement above is true then multiplication of two $n \times n$ matrices can be performed by a computation of size $O(n^2)$.

Unbounded Fan-in Circuits: We generalize the definition of a computation by allowing unbounded fan-in. That is, addition operations are any linear combination of the polynomials of the lower level and the multiplication operations can include any number of polynomials. We are interested in showing lower bounds for constant depth circuits (notice that such circuits are trivial without the unbounded fan-in assumption). When the depth is limited to 2, exponential lower bounds can be easily shown by considering an irreducible polynomial with an exponential number of monomials. Therefore, we consider depth 3 computations. Notice that we can transform any computation into a computation in which addition operations only use the results of multiplication operations and multiplication operations only use the results of addition operations. This increases the size of depth 3 computations by a polynomial factor. Thus, there are two possible types of computations. In a $\Pi\Sigma\Pi$ computation the first level includes multiplication operations, the second level includes addition operations and the last level includes one multiplication that provides the output. Lower bounds for $\Pi\Sigma\Pi$ computations can be shown by taking an irreducible polynomial with an exponential number of monomials and noticing that the last multiplication operation does not help. The second type of computation is $\Sigma\Pi\Sigma$ where the first level includes addition operations, the second level includes multiplication operations and the last level includes one addition that provides the output. This is the simplest non-trivial case and still, no exponential lower bounds are known:

Open Problem 3.6 *Find an explicit function which cannot be computed by a polynomial size* $\Sigma\Pi\Sigma$ *circuit with unbounded fan-in.*

A natural candidate for an exponential lower bound is the polynomial $Sym^d_n = \sum_{|S|=d, S\subseteq[n]} \Pi_{x\in S} x$. Over finite fields there exists an exponential lower bound on the

size of its calculation. Surprisingly, a result of Ben-Or shows that $S_{\Sigma\Pi\Sigma}(Sym_n^d) \leq n^2$ which means that for this polynomial the arithmetic model is stronger than the Boolean one. The idea is based on the polynomial $g(t) = \Pi_{i=1}^n(t+x_i) = \sum_{d=0}^n Sym_n^d(x)t^{n-d}$. Computing it can be done in an arithmetic circuit of depth two by adding t to each of the x_i's in parallel and then multiplying them all. We can also perform a depth 2 computation of $g(t_1), \ldots g(t_{n+1})$ for $n+1$ constants $t_1, \ldots, t_{n+1}$ by performing the same computation $n+1$ times in parallel. Now, by taking a linear combination of the polynomials $g(t_1), \ldots g(t_n)$ we can interpolate the coefficient of t^{n-d} which is exactly the output polynomial $Sym_n^d(x)$.

Ben-Or's computation uses intermediate polynomials of degree n although the final polynomial is of degree d. The following result of Nisan and Wigderson shows that this is necessary[1]:

Theorem 3.7 ([16]) *For computations whose intermediate polynomials are of degree at most d, $S_{\Sigma\Pi\Sigma}(Sym_n^d) \geq \binom{n}{d/2}2^{-d}$.*

This lower bound is exponential for, say, $d = n/8$ since $\binom{n}{n/16}2^{-n/8} \geq 16^{n/16}2^{-n/8} = 2^{n/8}$.

Proof Sketch: As mentioned earlier, we can assume that the computation is of type $\Sigma\Pi\Sigma$. We define the dimension $dimV$ of a set of polynomials V as the number of linearly independent polynomials in V. Also, for a set of variables $S = \{x_{i_1}, \ldots x_{i_k}\}$ let $\frac{\partial f}{\partial S}$ be $\frac{\partial f}{\partial x_{i_1}\partial x_{i_2}\ldots\partial x_{i_k}}$. The progress measure used to show this lower is the dimension of the set of all partial derivatives of a polynomial, that is, $\{\frac{\partial f}{\partial S} | S \subseteq X\}$. Denote this set by ∂f.

Claim 3.8 *The following two properties of the progress measure hold:*

- $dim(\partial \sum_i f_i) \leq \sum_i dim\partial f_i$
- $dim(\partial \Pi_i f_i) \leq \Pi_i dim\partial f_i$

Proof: The first property follows from the observation that $\partial \sum_i f_i$ is spanned by the union of ∂f_i for all i. The proof of the second property is as follows. Consider the product $\Pi_{i=1}^m f_i$ over the variables $X = \{x_1, \ldots, x_n\}$. Let $F_i \subseteq \partial f_i$ be a set of linearly independent polynomials, $|F_i| = dim\partial f_i$. Consider the set $F = \{g_1 \cdot \ldots \cdot g_n \mid g_i \in F_i\}$. Obviously, $|F| \leq \Pi_{i=1}^m |F_i| = \Pi_{i=1}^m dim\partial f_i$. We claim that any polynomial in ∂f is a linear combination of elements of F. Without loss of generality, consider $\frac{\partial f_1 \cdot \ldots \cdot f_m}{\partial S}$ where $S = \{x_1, \ldots, x_k\}$. It can be seen that,

$$\frac{\partial f_1 \cdot \ldots \cdot f_m}{\partial S} = \sum \frac{\partial f_1}{\partial S_1} \cdot \ldots \cdot \frac{\partial f_m}{\partial S_m}$$

[1] In their paper, they use a somewhat stronger assumption. A polynomial is said to be homogeneous when all its monomials are of the same degree; a computation is homogeneous when all the polynomials that appear in the computation are homogeneous. They assumed that the computation is homogenous. This was motivated by a result that shows that any computation can be turned into a homogenous computation with the increase of the depth by a factor of $O(\log d)$ where d is the degree of the polynomial computed (appears implicitly in [23], see [25]). This increase in the depth is conjectured to be necessary.

where the sum is over all the partitions of S into $S_1, \ldots, S_m$. By substituting each $\frac{\partial f_i}{\partial S_i}$ with a linear combination of elements of F_i we get that $\frac{\partial f_1 \cdot \ldots \cdot f_m}{\partial S}$ is a linear combination of elements of F. ■

We can now consider the behavior of this progress measure on a $\Sigma\Pi\Sigma$ circuit. The lowest level contains only variables and therefore the dimension is 2 ($\partial x_i = \{x_i, 1, 0\}$). After the addition operations of the lowest level, the dimension is still 2 because all the polynomials are sums of variables and therefore their partial derivatives are all 1 or 0 in addition to the polynomial itself. In the next level we multiply subsets of polynomials from the previous level. According to our assumption, the highest degree we encounter is not more than that of the output, i.e., not more than d. Therefore, the multiplications are of at most d polynomials and according to the second property of the progress measure, the dimension increases to at most 2^d. The last operation is an addition operation. Here, according to the first property, we must take at least $2^{-d} \cdot dim\partial Sym_n^d$ polynomials in order to reach the dimension of the output ($dim\partial Sym_n^d$).

Claim 3.9 $dim\partial Sym_n^d \geq \binom{n}{d/2}$.

Proof: Consider the $\binom{n}{d/2}$-vector v that includes all the partial derivatives $\frac{\partial Sym_n^d}{\partial S}$ for sets S of size $d/2$. By the definition of Sym_n^d, $\frac{\partial Sym_n^d}{\partial S}$ is exactly $Sym_{[n]-S}^{d/2}$ for a set S of size $d/2$. Consider the matrix $\binom{n}{d/2} \times \binom{n}{d/2}$ matrix A indexed by sets of size $d/2$ such that $A_{ST} = 1$ if $S \cap T = \phi$ and zero otherwise. This is a nonsingular matrix (e.g. [12, pp. 22-23]) and it can be seen that $Au = v$ where u is the $\binom{n}{d/2}$-vector that includes the monomial $\Pi_{x \in T} x$ in position T. Therefore, the dimension of v is $\binom{n}{d/2}$. ■

The completes the proof of Theorem 3.7. ■

Shpilka and Wigderson [20] show lower bounds without assuming that the intermediate polynomials are of degree at most d. For example, they show that $S_{\Sigma\Pi\Sigma}(Sym_n^d) \geq d(n-d)$. This is essentially tight since Ben-Or's result is an $\Sigma\Pi\Sigma$ circuit of size $O(n^2)$ for computing the symmetric polynomial. The idea is to note that in the above proof the only assumption we used is that the fan-in in the second level is bounded by d. On the other hand, we know that if the fan-in is large, the circuit must be large too. Hence, we can assume that the number of multiplication gates with large fan-in in not too large. Next, we eliminate the multiplication gate with large fan-in by zeroing one of their inputs. Since an input to a multiplication gate is a sum of variables, this can be done by satisfying an affine linear constraint. To complete the proof, one has to show that the dimension of ∂Sym_n^d remains high in the affine subspace to which we are restricted.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1975. Second printing, Addison-Wesley Series in Computer Science and Information Processing.

[2] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoret. Comput. Sci.*, 22(3):317–330, 1983.

[3] Allan Borodin and Ian Munro. *The computational complexity of algebraic and numeric problems*. American Elsevier Publishing Co., Inc., New York-London-Amsterdam, 1975. Elsevier Computer Science Library; Theory of Computation Series, No. 1.

[4] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic complexity theory*. Springer-Verlag, Berlin, 1997. With the collaboration of Thomas Lickteig.

[5] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.*, 19:297–301, 1965.

[6] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, 9(3):251–280, 1990.

[7] Rūsiņš Freivalds. Fast probabilistic algorithms. In *Mathematical foundations of computer science, 1979 (Proc. Eighth Sympos., Olomouc, 1979)*, pages 57–69. Springer, Berlin, 1979.

[8] Robin Hartshorne. *Algebraic geometry*. Springer-Verlag, New York, 1977. Graduate Texts in Mathematics, No. 52.

[9] Laurent Hyafil. On the parallel evaluation of multivariate polynomials. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing (San Diego, Calif., 1978)*, pages 193–195. ACM, New York, 1978.

[10] Mark Jerrum and Marc Snir. Some exact complexity results for straight-line computations over semirings. *J. Assoc. Comput. Mach.*, 29(3):874–897, 1982.

[11] E. Kaltofen and M. F. Singer. Size efficient parallel algebraic circuits for partial derivatives. In D. V. Shirkov, V. A. Rostovtsev, and V. P. Gerdt, editors, *IV International Conference on Computer Algebra in Physical Research*, pages 133–145, Singapore, 1991. World Scientific Publ. Co.

[12] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, Cambridge, 1997.

[13] J. Morgenstern. How to compute fast a function and all its derivatives, a variation on the theorem of Baur-Strassen. *Sigact News*, 16(4):60–62, 1985.

[14] Jacques Morgenstern. Note on a lower bound of the linear complexity of the fast Fourier transform. *J. Assoc. Comput. Mach.*, 20:305–306, 1973.

[15] Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 410–418, New Orleans, Louisiana, 6–8 May 1991.

[16] Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Comput. Complexity*, 6(3):217–234, 1996/97.

[17] Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. System Sci.*, 55(1, part 1):24–35, 1997. 26th Annual ACM Symposium on the Theory of Computing (STOC '94) (Montreal, PQ, 1994).

[18] Eli Shamir and Marc Snir. On the depth complexity of formulas. *Math. Systems Theory*, 13(4):301–322, 1979/80.

[19] Victor Shoup and Roman Smolensky. Lower bounds for polynomial evaluation and interpolation problems. *Comput. Complexity*, 6(4):301–311, 1996/97.

[20] Amir Shpilka and Avi Wigderson. Depth-3 arithmetic formulae over fields of characteristic zero. In *Fourteenth Annual IEEE Conference on Computational Complexity (Atlanta, GA, 1999)*, pages 87–96. IEEE Computer Soc., Los Alamitos, CA, 1999.

[21] Volker Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[22] Volker Strassen. Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. *Numer. Math.*, 20:238–251, 1972/73.

[23] Volker Strassen. Vermeidung von Divisionen. *J. Reine Angew. Math.*, 264:184–202, 1973.

[24] Volker Strassen. Algebraic complexity theory. In *Handbook of theoretical computer science, Vol. A*, pages 633–672. Elsevier, Amsterdam, 1990.

[25] L. G. Valiant. Negation can be exponentially powerful. *Theoret. Comput. Sci.*, 12(3):303–314, 1980.

[26] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983.

[27] Leslie G. Valiant. Why is Boolean complexity theory difficult? In *Boolean function complexity (Durham, 1990)*, pages 84–94. Cambridge Univ. Press, Cambridge, 1992.

[28] Joachim von zur Gathen. Algebraic complexity theory. In *Annual review of computer science, Vol. 3*, pages 317–347. Annual Reviews, Palo Alto, CA, 1988.